Big O notation is used to describe the efficiency of algorithms, especially how their runtime or memory usage grows relative to input size n.

Common Search Time Complexities:

Search Algorithm	Best Case	Average Case	Worst Case	Sorted Required
Linear Search	O(1)	O(n)	O(n)	No
Binary Search	O(1)	O(log n)	O(log n)	Yes

- Best Case: When the desired element is found in the first attempt.
- Average Case: General expectation over multiple inputs.
- Worst Case: When the element is not found or is the last one checked.

Linear Search:

- How it works: Scans each item in the list one by one until it finds the match.
- Advantages: Works on unsorted data.
- Disadvantages: Inefficient for large datasets.

Binary Search:

- How it works: Requires sorted data. Compares the target with the middle element and eliminates half of the array each step.
- Advantages: Much faster on large, sorted datasets.
- Disadvantages: Needs sorted input, and more logic to maintain sorting.

Time Complexity Comparison:

Criteria	Linear Search	Binary Search
Efficiency	Slower	Faster
Requires Sorting	No	Yes
Simplicity	Simple	More Complex
Use Case	Small/unsorted data	Large/sorted data

Which is better:

- For real-time searches on a large product catalog, binary search is preferred due to its logarithmic efficiency, but only if data is sorted.
- If the platform allows fuzzy, dynamic, or unsorted searching (like real-time filtering), then hashing or inverted indexing is even more optimal, though that is beyond basic search.